Combining SDL with Synchronous Data Flow Modelling for Distributed Control Systems*

Jean-Louis Camus and Thierry Le Sergent

Telelogic Technologies Toulouse
Jean-Louis.Camus@telelogic.com, Thierry.LeSergent@telelogic.com

Abstract. Engineers are faced nowadays with the challenge of designing strongly distributed control systems, with complex interactions. There is little theory and tool support to address this recent challenge. Control engineering and telecom engineering have dedicated but unrelated techniques, each for their specific domain. In this paper, we explore an approach where we combine two complementary formal methods, with good tool support and industrial acceptance: SCADE/Lustre from the Control Engineering domain, and SDL, from the Telecom domain.

1 Introduction

Traditional real time computerised systems were designed and implemented as fairly independent systems, having a strong interaction with the process they controlled, but very limited interaction with other computers. Also, they used to fall into specific categories with dedicated approaches, tools and techniques, such as:

- Programmable Logic Controllers, with focus on combinatorial or sequence logic;
- Regulation Controllers with focus on regulation laws;
- Telecom Systems (switches, phones) with focus on protocols.

For each of these categories, engineers have developed specific approaches, supported by dedicated notations and tools, such as SDL (Telecom), relay logic or SCADE (control engineering).

Things are getting more complex as the functions of computerised systems are becoming more and more sophisticated, and as these systems need to communicate via more complex protocols. As an example, new aircraft have to combine:

- Flight Control, involving both regulation and complex logic;
- Communication between embedded computers;
- Automatic communication with the environment (ground, satellites).

* Work supported by the European project CRISYS, EP 25.514 – Critical Instrumentation and control SYStems.

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 1–18, 2001.

Cars constitute another example, where synchronous, clock driven systems control the engine, gears, brakes, airbags, while at the other end of the scale asynchronous systems manage routes using GPS, and all computing units communicate through buses.

In this paper we show why it is not possible to design such systems by using just one of the above mentioned techniques. It is necessary to design systems using a global, consistent approach. Rather than reinventing the wheel, we prefer to rely on existing proven techniques, and use them in a sound way.

We introduce a formal technique, SCADE, which is suitable for the computer based, control engineering domain and show some advances of that technique in distributed control, and its limitations. We reiterate why SDL alone is generally not suited to the design of control system.

The contribution of this paper is to present a global approach, where we cleanly combine SDL and SCADE, and illustrate that approach with a simple example. Note: This approach is part of a larger project, where all notations are part of a UML framework (as is SDL-2000 already), but these UML aspects are beyond the scope of the current paper.

2 The SCADE/Lustre Technology

Telelogic Tau SCADE is the name for both the notation and the toolset maintained by Telelogic (`http://www.telelogic.com/SCADE`*). It is based on the formal language Lustre. We will call it SCADE in the remainder of this paper.*

2.1 Application Domain

Currently, SCADE is mostly used for the development of safety critical control systems: Airbus and Eurocopter embedded computers [1], Nuclear Power Plant control, Railway signalling systems [2], Car engine control.

2.2 The Language

The formal language Lustre is the basis of SCADE and was created to provide a simple, formal notation for the description of digital control systems. It is a bridge between control engineering and computer science. Lustre is a synchronous, data flow programming language [3]. It belongs to the family of synchronous programming languages, like Statecharts [4], Esterel [5] and SIGNAL [6].

A synchronous system computes its output from its input and state in "zero time": this does not mean that the real program takes zero real time, but rather that computation is completed before the output is communicated outside, and that there is no communication during the computation. In practice, this is generally realised by using sample and hold interfaces. Note that this property is relative to a clock and that several (sub-)systems may run at different speeds.